JBuddy Bot Framework™ Users Guide

Version 2.0

2008 - 2012

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

1 Introduction

Ever since the dawn of computing, humans have been supplying input to computers and computers have been responding to human input. Enter Instant Messaging, or IM as it's well known: humans communicating in real time to other humans using text messages. Where is the human input and computer responses in IM? Actually it has been a growing trend in IM and already has a name - IM Bots. In IM, a Bot is a non-human participant in the IM network. Bots can be thought of as just another buddy in your buddy list, however they are not human.

In May 2007, we released JBuddy Bot Framework version 1.0 and gave software developers worldwide, a taste at how easy developing an IM Bot could be. Since then, we've been hard at work on the next release. It's finally here - 2.0! One thing you'll notice right away, we didn't hold back on innovation and new features. Be sure to checkout new sections of this guide covering the new bot action types for URLs including file and http and the new Routes feature. Last but certainly not least, you'll notice a new section called Custom Client Gateways, which discusses how the JBuddy Bot Framework can be extended to non-IM messaging systems including SMS, etc, opening up a whole new world for interactive message application development.

Without further ado, let's get started!

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

2 Getting Started

2.1 The Bot Definition File

The bot engine processes messages and events from users and sends back responses. The engine is driven by an XML file called the Bot Definition File.

The Bot Definition File contains markup defining the bot's logic and content. It contains a top-level <bot> element which looks like this:

```
<bot xmlns="http://www.zionsoftware.com/jbuddy/bot">
    <!-- logic and content goes here -->
</bot>
```

All elements in the Bot Definition File are part of the "http://www.zionsoftware.com/jbuddy/bot" namespace.

2.2 Running a Bot

To run a bot, use the Run utility.

The Run utility allows you to run a bot in the background (daemon or service). By default, it also connects Client Gateways, if available (see the section on "Client Gateways" for more information).

Assuming your Bot Definition File is named "bot.xml", type the following at the command line from the lib folder of the installation directory:

```
java -cp Bots.jar Run
```

If your bot definition file is not named "bot.xml", use the -file or -f argument followed by the file path. For example, for a file named "mybot.xml", type the following at the command line:

```
java -cp Bots.jar Run -file mybot.xml
```

2.3 Testing a Bot

To test a bot, use the Test utility.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

With the Test utility, you can issue commands to the bot from the command line. By default, Test mode does not connect Client Gateways. Instead, Test mode creates a local-only test client that takes input from the command line (stdin). You can use the Test utility to test most of the examples in this guide.

Assuming your Bot Definition File is named "bot.xml", type the following at the command line from the lib folder of the installation directory:

```
java -cp Bots.jar Test
```

If your bot definition file is not named "bot.xml", use the -file or -f argument followed by the file path. For example, for a file named "mybot.xml", type the following at the command line:

```
java -cp Bots.jar Test -file mybot.xml
```

2.4 Debugging a Bot

To help understand what the bot engine is doing and to help fix potential problems, you can enable debug logging. This is done by setting the bot's log level. Here, we set the log level to "all", which prints all log messages to the console, regardless of severity:

```
<bot xmlns="http://www.zionsoftware.com/jbuddy/bot" logLevel="all">
    <!-- logic and content goes here -->
</bot>
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

3 Client Gateways

Client Gateways allow a bot to connect to external networks, such as public or private Instant Messaging servers. Once a bot is connected to gateways, users on the associated networks may interact with the bot.

By default, the Run utility connects all client gateways at start and the Test utility does not. You force client gateways to connect by including the “-connect true” arguments to Run or Test or prevent the Bot engine from connecting client gateways by including the “-connect false” arguments to Run or Test.

3.1 Connecting a Bot to IM Networks

To connect a bot to an IM network, use the <client> element in the Bot Definition File. It must be defined under the top level <bot> element. The following example connects the bot to the AOL Instant Messenger network:

```
<client protocol="AIM" name="myaimbot" password="mypassword"/>
```

(Replace "myaimbot" and "mypassword" with your AIM login information). Once signed in to AIM, AIM users may send messages to, and interact with, the bot. Multiple gateways can be defined, to connect the bot to several networks simultaneously. Here are the available network protocols, as provided by the JBuddy SDK:

Public IM:

- AIM - AOL Instant Messenger
- ICQ - "I Seek You"
- MSN - MSN Messenger (also known as Windows Live Messenger)
- YIM - Yahoo Messenger

Enterprise IM:

- JABBER - Jabber/XMPP
- LCS - Microsoft’s Office Live Communications Server
- SAMETIME - IBM Lotus Sametime
- JSC - Zion Software's JBuddy Message Server

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

You may use the Run utility to run your bot and connect it to the Client Gateways you defined.

3.2 Client Properties

Special properties can be set in a client definition. These can be used to override similar entries in the JBuddy SDK's protocol properties files, such as "AIM.properties." For example, we can add a property to tell JBuddy to send a keep-alive message to the server every 60 seconds from our AIM client:

```
<client protocol="AIM" name="myaimbot" password="mypassword">
    <property name="JBUDDY_AIM_KEEP_ALIVE_INTERVAL" value="60"/>
</client>
```

3.3 Password Security

As a security measure, to avoid using plain-text passwords in a bot definition, passwords can be encrypted.

3.3.1 The EncryptPassword Utility

The utility asks for a password and an optional "pass key." It returns an encrypted password string, which may be used in a client definition.   Type the following at the command line, from the lib folder of the installation directory:

```
java -cp Bots.jar EncryptPassword
```

You are prompted to enter some values. Here's an example session:

```
Enter a password: mypassword
Enter a pass key (optional):
The encrypted value is: 5xnFMAERlD0jdfnP5WeA/g==
```

Now, place the encrypted value in the client definition:

```
<client protocol="AIM" name="myaimbot" password="5xnFMAERlD0jdfnP5WeA/g=="
passwordEncrypted="true"/>
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

Setting passwordEncrypted to "true" tells the bot engine that the password is encrypted, and needs to be decrypted in order to connect.

3.3.2 Using Pass Keys

Password security can be increased further by introducing a custom pass key that is used to encrypt the client passwords. Here is an example EncryptPassword session with a custom pass key:

```
Enter a password: mypassword
Enter a pass key (optional): mypasskey
The encrypted value is: 24Qw020G/TzpZdKOrnnydg==
```

Note that the encrypted value is now different from the earlier example, which did NOT use a pass key. Place the encrypted value in the client definition as usual:

```
<client protocol="AIM" name="myaimbot" password="24Qw020G/TzpZdKOrnnydg=="
passwordEncrypted="true"/>
```

You must specify the custom pass key to the Run utility in order for the engine to properly decrypt passwords. To run the previous example with the new pass key, type the following at the command line:

```
java -cp Bots.jar Run -passkey mypasskey
```

3.4 Custom Client Gateways

The Bot Framework is extensible, allowing you to connect a bot to networks not already supported by the JBuddy SDK. The Java class "com.zion.jbuddy.bots.BotGateway" is provided, and can be extended to create a custom Client Gateway implementation. The following example might be used to add SMS (mobile text message) support to a bot, where the fictional "com.acme.bots.SMSGateway" class extends the BotGateway class:

```
<client protocol="SMS" gatewayClass="com.acme.bots.SMSGateway">
    <property name="shortCode" value="12345"/>
</client>
```

For more information, see the javadocs provided with the framework.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

4 Targets

Targets are tasks carried out by a bot in response to an event, such as a message from a user. They can be used to send messages or files to the user, carry out actions, and more. A target is defined under the <bot> element, like this:

```
<target command="SOME_COMMAND">
    <title>SOME_TITLE</title>
    <!-- define messages, files, etc here -->
</target>
```

The command is a one-word command issued by a user (human or non-human) which triggers this target. The title is used to describe this target within menus (see the section on menus for more information).

4.1 Sending Content and Messages

In this example, we have an "about" target that provides some simple content to the user:

```
<target command="about">
    <title>About This Bot</title>
    <content>This is a test bot built by <i>Zion Software</i>
      to show the capabilities of the
      <i>JBuddy Bot Framework</i>.</content>
</target>
```

Now, when a user types "about", the content ("This is a test bot...") is sent to them in the form of an instant message. Content may contain HTML-like markup which is parsed by the bot engine and shown as rich text. Supported HTML tags include <b> (bold), <i> (italic), <u> (underline), <s> (strike-through), <font>, <sup> (superscript), <sub> (sub-script), and <a> (hyperlink). A <message> element can also be used to send a mes-sage. The previous example can be rewritten as follows:

```
<target command="about">
    <title>About This Bot</title>
    <message type="IM">This is a test bot built by
      <i>Zion Software</i> to show the capabilities of the
      <i>JBuddy Bot Framework</i>.</message>
</target>
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

Recipients can also be specified in a message, using a comma-delimited list of user names. The above example can be modified to send the "about" info to users other than the one who typed the command (in this case, to both "adam" and "joe"):

```
<target command="about">
    <title>About This Bot</title>
    <message type="IM" recipients="adam,joe">This is a test bot built by
      <i>Zion Software</i> to show the capabilities of the
      <i>JBuddy Bot Framework</i>.</message>
</target>
```

Note the differences between <content> and <message>. <content> is used to send simple instant message replies. Multiple, successive <content> elements are combined into a single instant message to the user, and recipients cannot be specified.

4.2 Using Parameters

Parameters can be used to ask a user for input. The user's input can be used later by the engine in various ways.

4.2.1 Getting User Input

The following example asks the user for his or her birth date:

```
<target command="birth">
    <title>Enter Your Birth Date</title>
    <parameter name="date">
      <description>Please enter your birth date (MM/DD/YYYY):</description>
    </parameter>
    <content>You entered: ${date}</content>
</target>
```

If a user types "birth", the parameter's description is displayed to the user. The value entered by the user is saved as the "date" parameter value. A message is then sent back, confirming the date the user entered. (${date} is an example of a content variable. The engine replaces ${date} with the value of the "date" parameter).

Here is an example session:

```
User: birth
 Bot: Please enter your birth date (MM/DD/YYYY):
User: 04/15/1980
 Bot: You entered: 04/15/1980
```

The user can also choose not to enter a value, canceling execution of the parameter (and its target) by simply typing "cancel."

4.2.2 Validating User Input (with Patterns)

The previous example can be improved to validate the user's input, only accepting a value from the user if it matches a specified pattern:

```
<target command="birth">
    <title>Enter Your Birth Date</title>
    <parameter name="date" pattern="\d{2}/\d{2}/\d{4}">
      <description>Please enter your birth date (MM/DD/YYYY):</description>
    </parameter>
    <content>You entered: ${date}</content>
</target>
```

The pattern is a regular expression. It matches 2 digits (the month), followed by a forward slash, followed by 2 digits (the day), followed by a forward slash, followed by 4 digits (the year).

Here is an example session:

```
User: birth
 Bot: Please enter your birth date (MM/DD/YYYY):
User: april 15 1980
 Bot: Invalid input.
      Please enter your birth date (MM/DD/YYYY):
User: 04/15/1980
 Bot: You entered: 04/15/1980
```

4.3 Sending Files

Files can also be sent to the user. The following example creates a "file" target that sends a file named "test.gif":

```
<target command="file">
```

```
    <title>Request a File</title>
    <file path="test.gif">
      <description>Hey ${displayName},
     Please accept this image.</description>
    </file>
</target>
```

${displayName} is another example of a content variable. The engine replaces this variable with the value of the user's display name. Also, like messages, recipients can be specified:

```
<target command="file">
    <title>Request a File</title>
    <file path="test.gif" recipients="adam,joe">
      <description>Hey ${displayName},
          Please accept this image.</description>
    </file>
</target>
```

4.4 Nested Targets

Targets can call other targets.

Consider the following target:

```
<target command="nested">
    <content>Here is some content.</content>
    <target ref="about"/>
    <content>Here is some more content.</content>
</target>
```

If a user submits "nested", the bot sends the following message:

```
Here is some content.
This is a test bot built by <i>Zion Software</i>
    to show the capabilities of the
    <i>JBuddy Bot Framework</i>.
Here is some more content.
```

Note the use of the "ref" attribute. It is used to reference the existing "about" target by its command name.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

5 Actions

Actions are external processes carried out by the bot engine. They can be used to execute HTTP (web) requests, call Java-based tasks, execute a system call, or more.

5.1 URL Actions

URL actions process content from a local or fully qualified URL (Uniform Resource Locator). The bot engine supports URL protocols provided by the underlying Java Runtime Environment (JRE), such as “file”, “http”, “https”, and more.

5.1.1 Files

The bot engine can retrieve content from a local file.

Here is an example:

```
<target command="about">
    <action type="url" path="about.html"/>
</target>
```

If a user submits "about", the contents of the local file "about.html" are parsed and sent to the user. The bot engine understands many basic HTML tags, such as styles, bold, font, and more.

5.1.2 HTTP Requests

HTTP requests can be used to retrieve web content or invoke web services. Here’s an example of a URL action using HTTP:

Here's an example of an HTTP action:

```
<target command="navaltime">
    <title>View the Current USNO Master Time</title>
          <action type="url"
path="http://tycho.usno.navy.mil/cgi-bin/timer.pl"/>
</target>
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

If a user submits "navaltime", the contents of

<http://tycho.usno.navy.mil/cgi-bin/timer.pl> are parsed as HTML and sent to the user.

The HTTP method can be set to “get” or “post”, among other options.

If parameters are specified in the target, the name/value pair are sent within the request. For GET requests, they are embedded in the URL’s query string. For other requests such as POST, they are sent as “application/x-www-form-urlencoded” content.

5.2 Java Actions

A Java action loads and executes an external Java class. The class must extend the abstract com.zion.jbuddy.bots.BotActionTask class. Here's an example of a Java action:

```
<target command="java">
    <title>A Java Action</title>
    <parameter name="firstname">
      <description>Enter your first name:</description>
    </parameter>
    <action type="java" path="MyActionTask"/>
</target>
```

Here is the code for MyActionTask.java:

```
import java.util.*;
import com.zion.jbuddy.bots.*;

/**
 * An example action task.
 */
public class MyActionTask extends BotActionTask {

    public Object execute() {
      return "Hi there, " + parameters.get("firstname");
    }
}
```

Parameter values are saved in the parameters Map attached to the BotActionTask. The content returned from the execute method is sent to the user as an instant message, similar to a <content> element. Here is an example session:

```
User: java
 Bot: Enter your first name:
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

```
User: Bob
 Bot: Hi there, Bob!
```

5.3 System Actions

A system action calls a system command or executable file. The results of the command are returned to the user. Here's an example of a system action:

```
<target command="ping">
    <title>Ping an Address</title>
    <parameter name="address">
      <description>Please enter an address to ping.</description>
    </parameter>
    <action type="system" path="ping ${address}"/>
</target>
```

If a user types "ping", the bot asks the user to enter a value for "address.” After the user enters the address, the system "ping" command is called. The address value is passed to the ping command (the content variable ${address} is replaced with the parameter's value). Finally, the results of the ping command are sent to the user in an instant message.The results could also be saved to a file, which is then sent to the user:

```
<target command="ping">
    <title>Ping an Address</title>
    <parameter name="address">
      <description>Please enter an address to ping.</description>
    </parameter>
    <action type="system" path="ping ${address}" resultType="file"/>
</target>
```

The "resultType" attribute dictates what type of content is sent to the user (in this case, a file).

5.4 XSL Transformations

XSL Transformations (XSLT) can be used to dynamically transform content. An XSLT template can be defined (using pure XML) that transforms XML or HTML input into another form.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

Consider the following example:

```
<target command="weather">
    <action type="url" path="http://www.somehost.com/current_weather.html"
transform="weather.xsl" resultType="xhtml"/>
</target>
```

If a user types "weather", information about the current weather conditions is retrieved from <http://www.somehost.com/current_weather.html>. The HTML content is transformed using the local "weather.xsl" template. The XSLT output is then parsed as XHTML content and sent to the user.

Most HTML content (especially content on the web) is not valid XML. To rectify this, the bot engine knows how to convert HTML content into valid XML if necessary (using HTML Tidy), so it can be processed using XSLT.

For more information on XSLT, see <http://www.w3.org/TR/xslt>, or the demos provided with the framework.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

6 Menus

Menus are used to organize targets (and other menus) into a menu hierarchy. When a user opens a menu, its available targets and sub-menus are presented to the user. Menus are defined under the top-level <bot> element. Here is an example of a simple menu:

```
<menu command="options">
    <title>Options Menu</title>
    <description>Here are the available options.</description>
    <target command="ping">
      <title>Ping an Address</title>
      <parameter name="address">
          <description>Please enter an address to ping.</description>
      </parameter>
      <action type="system" path="ping ${address}" resultType="file"/>
    </target>
    <target command="about">
      <title>About This Bot</title>
      <content>This is a test bot built by
          <i>Zion Software</i> to show the capabilities of the
          <i>JBuddy Bot Framework</i>.</content>
    </target>
</menu>
```

Here is an example session:

```
User: options
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
```

The menu displays its available target commands and their descriptive titles. The targets from earlier examples are now available within this menu. After loading this menu, the user can use the "about" and "ping" commands. Here is an example session:

```
User: options
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
User: ping
 Bot: Please enter an address to ping.
```

Remember that targets can also be referenced from other targets. The previous example can be rewritten using references:

```
<menu command="options">
    <title>Options Menu</title>
    <description>Here are the available options.</description>
    <target ref="ping"/>
    <target ref="about"/>
</menu>
```

For the references to work, the "ping" and "about" targets must be defined under <bot>, like in the earlier examples.

6.1 Creating Sub-Menus

Menus can be defined under other menus, creating sub-menus. The following example adds an "actions" sub-menu to "options":

```
<menu command="options">
    <title>Options Menu</title>
    <description>Here are the available options.</description>
    <target ref="ping"/>
    <target ref="about"/>
    <menu command="actions">
      <title>Actions Menu</title>
      <description>Here are the available actions.</description>
      <target ref="java"/>
      <target ref="system"/>
    </menu>
</menu>
```

Like targets, menus can also be referenced. The above example can be rewritten using references. In this example, "actions" is a top-level menu, but can also be found as a sub-menu of "options."

```
<menu command="actions">
    <title>Actions Menu</title>
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

```
    <description>Here are the available actions.</description>
    <target ref="java"/>
    <target ref="system"/>
</menu>

<menu command="options">
    <title>Options Menu</title>
    <description>Here are the available options.</description>
    <target ref="ping"/>
    <target ref="about"/>
    <menu ref="actions"/>
</menu>
```

6.2 Navigating a Menu Hierarchy

Using the earlier examples, the user may navigate the menu hierarchy. Here is an example session:

```
User: options
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
      actions - Actions Menu
User: actions
 Bot: Actions Menu

      Here are the available actions.

      java - A Java Action
      system - A System Action
      back - Back to Options Menu
```

Note that the "actions" sub-menu contains a selection that wasn't defined before: "back.” The "back" command is automatically available when a user navigates to a new menu while viewing another. Here is an example session, illustrating the "back" command:

```
User: options
 Bot: Options Menu

      Here are the available options.
      ping - Ping an Address
      about - About This Bot
      actions - Actions Menu
User: actions
 Bot: Actions Menu

      Here are the available actions.

      java - A Java Action
      system - A System Action
      back - Back to Options Menu
User: back
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
      actions - Actions Menu
```

Another built-in command, "refresh", can be used to show the last menu the user navigated to:

```
User: options
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
      actions - Actions Menu
User: refresh
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
      actions - Actions Menu
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

7 Event Handlers

Event handlers are similar to targets, but instead of being triggered by user commands, they are triggered in response to certain events, such as a user message or status change.The <eventHandler> element is defined under the <bot> element, just like targets and menus. In the following example, an event handler is triggered when a user sends a message to the bot:

```
<eventHandler type="message">
    <content>Hey! You just sent me a message.</content>
</eventHandler>
```

Note that the message may be an IM, a typing notification, or more. Event handlers can also be called in response to presence updates (such as when a user changes their status):

```
<eventHandler type="presence">
    <content>I just received your presence update!</message>
</eventHandler>
```

Now, when the bot receives a presence update from a user on its buddy list, it sends them an instant message.

7.1 Menu-Local Event Handlers

Event handlers can be defined directly within menus. A menu-local event handler can only be triggered when the user is viewing the menu it is contained in. For example, we can define a menu-local event handler in the "options" menu we defined earlier:

```
<menu command="options">
    <title>Options Menu</title>
    <description>Here are the available options.</description>
    <target ref="ping"/>
    <target ref="about"/>
    <menu ref="actions"/>
    <eventHandler type="message">
      <content>Hey! You just sent Options Menu a message.</content>
    </eventHandler>
</menu>
```

Now, if a user types "options", and then types any message, the bot will send a message back. Here is an example session:

```
User: options
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

```
 Bot: Options Menu

      Here are the available options.

      ping - Ping an Address
      about - About This Bot
      actions - Actions Menu
User: hi
 Bot: Hey! You just sent Options Menu a message.
```

7.2 Consuming Events

Normally, when an event occurs, the bot engine will try to trigger each associated event handler, in order. If you'd like an event handler to prevent the engine from triggering more event handlers for the same event, you can "consume" the event. Take the following example:

```
<eventHandler type="message" consume="true">
    <content>Hello there!</content>
</eventHandler>

<eventHandler type="message">
    <content>Hello there! (You won't see this!)</content>
</eventHandler>
```

If a user types a message, the bot triggers the first event handler, but since the event is consumed there, the second event handler is NOT triggered. The engine processes event handlers first, and then if the event was an IM message, it processes user commands. Therefore if an event is consumed, it will not trigger any further event handlers, targets, or menus.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

8 Filters

Filters can be used to restrict a <target>, <menu>, or <eventHandler> to certain users or properties. Each of these elements may contain a <filters> element that represents a set of filters.

8.1 Include Filters

An include filter allows only users or events that match the filter. If any include filters are defined, at least one of them must match for the target or event handler to be triggered. For example, an include filter could be used to create an event handler that responds to IM type messages, ignoring all other types of messages:

```
<eventHandler type="message">
    <filters>
      <include messageType="IM"/>
    </filters>
    <content>Hey! You just sent me an IM.</content>
</eventHandler>
```

A more powerful example is an event handler that automatically accepts buddy authorization requests on the Yahoo Messenger network:

```
<eventHandler type="message">
    <filters>
      <include protocol="YIM" messageType="AUTH_REQUEST"/>
    </filters>
    <message type="AUTH_ACCEPT"/>
    <content>Thanks for adding me to your buddy list!</content>
</eventHandler>
```

Filters may apply to presence event handlers. The following example sends a message to users on the bot's buddy list when they are BUSY:

```
<eventHandler type="presence">
    <filters>
      <include status="BUSY"/>
    </filters>
    <content>I can see that you are busy!</content>
</event>
```

Filters apply to targets and menus, too. The following example creates an "about" target that is only available to users using the JABBER protocol:

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

```
<target command="about">
    <title>About Jabber</title>
    <filters>
      <include protocol="JABBER"/>
    </filters>
    <content>You are using the Jabber protocol!</content>
</target>
```

You can also define multiple similar targets with separate filters. The first target that is allowed will be triggered. The following example defines an "about" command that returns a different target depending if a user is on AOL Instant Messenger, MSN Messenger, or Yahoo Messenger:

```
<target command="about">
    <title>About AOL Instant Messenger</title>
    <filters>
      <include protocol="AIM"/>
    </filters>
    <content>You are using AOL Instant Messenger!</content>
</target>

<target command="about">
    <title>About MSN Messenger</title>
    <filters>
      <include protocol="MSN"/>
    </filters>
    <content>You are using MSN Messenger!</content>
</target>

<target command="about">
    <title>About Yahoo Messenger</title>
    <filters>
      <include protocol="YIM"/>
    </filters>
    <content>You are using Yahoo Messenger!</content>
</target>
```

8.2 Exclude Filters

An exclude filter tells the engine NOT to allow users or events that match the filter. The following example sends a response message to users, but only if they are NOT using the JABBER protocol:

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

```
<eventHandler type="message">
    <filters>
      <include messageType="IM"/>
      <exclude protocol="JABBER"/>
    </filters>
    <content>Hey! You just sent me an IM.</content>
</eventHandler>
```

We could also hide the actions menu from JABBER users:

```
<menu command="actions">
    <title>Actions Menu</title>
    <description>Here are the available actions.</description>
    <filters>
      <exclude protocol="JABBER"/>
    </filters>
    <target ref="java"/>
    <target ref="system"/>
</menu>
```

8.3 Using Patterns In Filters

Powerful regular expression patterns can be used anywhere in a filter. For example, we can make the bot send a response message to users on the "jabber.org" Jabber server only:

```
<eventHandler type="message">
    <filters>
      <include messageType="IM" protocol="JABBER" user=".*@jabber\.org"/>
    </filters>
    <content>Hey! You just sent me an IM.</content>
</eventHandler>
```

Note the value of the "user" attribute. It is a regular expression meaning (one or more characters followed by "@jabber.org").

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

9 Routes

Routes are used to direct responses to other users or client gateways.

9.1 Using Routes

A single route represents a sender (client) and recipients (users) to direct responses to. One or more routes can be specified on a target, menu, or event handler. In this example, we create a simple route:

```
<target command="test">
    <routes>
        <route recipients="someuser,someotheruser"/>
    </routes>
    <content>Hello!</content>
</target>
```

If a user submits "test", the bot sends a "Hello!" message to "someuser" and "someotheruser", using the same gateway as the initial user.

9.2 Routes with Client Gateways

Routes can also be used to send responses to users on other gateways. Each gateway is assigned an ID, which can be used to specify a route sender. By default, the ID is equal to "name#protocol", where "name" is the client's name and "protocol" is the protocol being used. The following gateway's ID is "myaimbot#AIM":

```
<client protocol="AIM" name="myaimbot" password="mypassword"/>
```

The simple route example can then be modified to send to users on AIM:

```
<target command="test">
    <routes>
      <route sender="myaimbot#AIM" recipients="someuser,someotheruser"/>
    </routes>
    <content>Hello!</content>
</target>
```

A custom ID can also be specified using the "id" attribute:

```
<client protocol="AIM" name="myaimbot" password="mypassword" id="aim"/>
```

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

10 Dynamic Content

Dynamic features can be added to a bot.

10.1 Dynamic XML

Raw bot definition elements can be returned from an action. The XML is parsed and processed by the engine on-the-fly, thus creating dynamic content. In this example, we create a Java action that returns a menu using dynamic XML. Here is the target that calls the Java action:

```
<target command="dynamicmenu">
    <title>Create a Dynamic Menu</title>
    <action type="java" path="DynamicMenuTask" resultType="xml"/>
</target>
```

Setting the action resultType to "xml" tells the engine that the action returns dynamic XML that needs to be processed. Here is the source code for "DynamicMenuTask.java":

```
import com.zion.jbuddy.bots.*;

public class DynamicMenuTask extends BotActionTask {

    public Object execute() {
      return "<menu xmlns='http://www.zionsoftware.com/jbuddy/bot'>"
           + "  <title>A Dynamic Menu</title>"
           + "  <menu ref='options'/>"
           + "  <target ref='about'/>"
           + "</menu>";
    }
}
```

Note that all features available in the bot definition XML are available dynamically, including target and menu references.

10.2 The Java API

A complete Java API is included with the framework that can also be used to create dynamic content.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

We can rewrite the previous example's source code to build the dynamic menu using pure Java code, instead of XML:

```
import com.zion.jbuddy.bots.*;
import com.zion.jbuddy.richcontent.RichContent;

public class DynamicMenuTask extends BotActionTask {

    public Object execute() {
      BotMenu menu = new BotMenu();
      RichContent titleContent = new RichContent();
      titleContent.append("A Dynamic Menu");
      menu.setTitle(titleContent);

      BotEngine engine = user.getClient().getEngine();

      BotTarget optionsMenu = engine.getTargets().getByID("options");
      menu.getTargets().add(optionsMenu);

      BotTarget aboutTarget = engine.getTargets().getByID("about");
      menu.getTargets().add(aboutTarget);
      return menu;
    }
}
```

For more information on using the Java API, consult the related examples and the javadocs included with the framework.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

11 Available Content Variables

These are the content variables that can be used within element content and certain attributes.

11.1 User

These variables apply to the user.

`${protocol}`
The user's protocol.

`${name}`
The user's name.

`${displayName}`
The user's display name.

`${command}`
The command the user entered (if any).

`${commandValue}`
The command value entered by the user (if any). The command value is everything entered after the command name.

`${messageType}`
The type of message the user entered (if any).

`${message}`
The message contents the user entered (if any).

`${status}`
The user's status.

`${statusMessage}`
The user's status message.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

11.2 Client

These variables apply to the user's associated client gateway.

`${client.protocol}`
The client's protocol.

`${client.name}`
The client's name.

`${client.displayName}`
The client's display name.

`${client.id}`
The client's ID.

`${client.status}`
The client's status.

`${client.statusMessage}`
The client's status message.

11.3 Statistics

These variables apply to the bot engine.

`${statistics.sessions}`
The current number of user sessions.

`${statistics.peakSessions}`
The peak (highest) number of concurrent user sessions that occurred since the engine was started.

`${statistics.totalSessions}`
The total number of user sessions handled since the engine was started.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.

`${statistics.uniqueSessions}`
The number of unique user sessions handled in the last 24 hours since the engine was started.

`${statistics.sentMessages}`
The number of messages sent since the engine was started.

`${statistics.receivedMessages}`
The number of messages received since the engine was started.

11.4 Client Statistics

These variables apply to the user's associated client gateway.

`${client.statistics.sessions}`
The current number of user sessions.

`${client.statistics.peakSessions}`
The peak (highest) number of concurrent user sessions that occurred since the engine was started.

`${client.statistics.totalSessions}`
The total number of user sessions handled since the engine was started.

`${client.statistics.uniqueSessions}`
The number of unique user sessions handled in the last 24 hours since the engine was started.

`${client.statistics.sentMessages}`
The number of messages sent since the engine was started.

`${client.statistics.receivedMessages}`
The number of messages received since the engine was started.

Copyright 2007-2012, Zion Software, LLC. All Rights Reserved.